Identifying Useful Features for Malware Detection in the Ember Dataset

Yoshihiro Oyama
University of Tsukuba
oyama@cs.tsukuba.ac.jp

Takumi Miyashita
University of Tsukuba
takumi.miyashita@syssec.cs.tsukuba.ac.jp

Hirotaka Kokubo
FUJITSU LABORATORIES LTD.
kokubo.hirotaka@jp.fujitsu.com

***Abstract*—Many studies have been conducted to detect malware based on machine learning of program features extracted using static analysis. In this study, we consider the task of distinguishing between malware and benign programs by learning their surface features, such as general file information and imported functions. To make such attempts practical, a good balance among accuracy, learning time, and feature-data sizes is required. Although using only a subset of features can reduce the required time and data sizes, it is not trivial to select an appropriate subset of features. In this paper, we present a case study of feature selection in malware detection based on supervised machine learning. We used the Ember dataset as the target data and identified useful combinations of features in terms of accuracy, learning time, and data size.**

***Index Terms*—malware detection, machine learning, feature selection, static analysis**

I. Introduction

Hundreds of millions of new malware programs appear each year and hence malware detection based on signature matching is becoming increasingly impractical. Machine learning (ML) is a promising countermeasure against modern malware because it provides the potential to detect malware programs without specific signatures of their data or behavior.

Generally, ML-based malware detectors extract the *features* of malware programs (and also benign programs, ideally) using static and/or dynamic analysis. They then learn the features to create *models* for determining whether a given program is malware. The models are used to estimate the likelihood that a given unknown program is malware.

One frequently occurring problem in ML-based malware detection is the great amount of time required to learn features of programs when the number or size of features is large or the number of programs is large. Although reducing the number of features shortens the learning time, it is also likely to decrease the detection accuracy. Although a combination of well-selected features may be able to simultaneously achieve acceptable accuracy, a short learning time, and modest data size, it is not trivial to select them.

In this study, we evaluate individual features of programs in terms of the accuracy of ML-based malware detection, and the temporal and spatial cost of creating models for it. We particularly consider surface features extracted using static analysis, such as file sizes, byte occurrence histograms, section information, and imported functions. The target data are contained in the Ember dataset [1]. We identify the best combinations of feature groups in the dataset that achieve a good balance between cost and detection accuracy. We also present the best combinations of feature groups and the best singleton feature groups in terms of detection accuracy alone. This study contributes to a better understanding of the usefulness of individual surface features and their combinations.

II. Ember Dataset

A. Data Included

The Ember dataset [1] has been public since April 2018 and contains surface feature data of more than one million samples of malware and benign software. The dataset consists of a set of JSON files, each of which represents the features of a malware sample. The dataset includes eight groups of surface features, which are shown in Table I. A combination of feature values extracted from a single program compose a feature vector. Each feature vector is associated with three items of extra information. The first item is the SHA256 hash of the program file. The second item is the month and year of the collection of the program. The final item is the "ground truth," which indicates the correct label of the program file, that is, "malicious," "benign," or "unlabeled."

Fig. 1 shows an extract of a JSON file of the dataset, which contains three items of extra information and feature values. Value 1 of the `label` item indicates "malicious," whereas value 0 indicates "benign." The feature value for each group is labeled with the name of its group.

We briefly explain the feature groups; their details are provided in the original paper [1].

- General file information (general): general file information, such as the file size; number of imported functions, exported functions, and symbols; and the presence of a debug section, resources, or signature.
- Header information (header): COFF header and original header information, such as timestamps, target machine strings, file magic, image versions, linker versions, and commit sizes.
- Imported functions (imports): information on pairs of imported functions and their source libraries.
- Exported functions (exports): list of exported functions.
- Section information (section): properties of each section, such as the name, size, virtual size, and list of strings that represent the section characteristics.

978-1-7281-5268-4/19/$31.00 ©2019 IEEE
DOI 10.1109/CANDARW.2019.00069

TABLE I: Feature Groups Included in the Ember Dataset

Feature group	Number of features
General file information (general)	10
Header information (header)	62
Imported functions (imports)	1,280
Exported functions (exports)	128
Section information (section)	255
Byte histogram (histogram)	256
Byte-entropy histogram (byteentropy)	256
String information (strings)	104
All	2,351

```
{
  "sha256": "04637...", "appeared": "2017-01",
  "label": 1,
  "histogram": [ 3818, 155, 135, ... ],
  "byteentropy": [ 0, 0, 0, ... ],
  "strings": { "numstrings": 170,
               "avlength": 8.170588235294117,
               ... },
  "general": { "size": 33334, "vsize": 45056,
               "has_debug": 0, ... },
  ...
}
```

Fig. 1: Example of Feature Data in the Dataset

- Byte histogram (histogram): number of occurrences of each byte value within the file. The 256 byte values in the histogram are normalized to a distribution using the file size. Generally, a byte distribution of a program file is closely related to whether the program is malware because a highly uniform distribution is observed in packed or obfuscated program files.
- Byte-entropy histogram (byteentropy): information on the byte-entropy histogram that approximates the joint distribution of the entropy and byte value.
- String information (strings): number of strings, their average length, and a histogram of the printable characters in the strings, and the entropy of characters across all printable strings. It contains the number of strings that are likely to be a path, URL, registry key, and magic number of executable files.

Table II shows the number of samples included in the dataset. The dataset is composed of 900,000 training data items contained in six files (`train0`, ..., `train5`) and 200,000 test data items contained in one file (`test`). Each training data file contains sample files collected in a distinct time period. The malware samples in the test data are all newer than those in the training data. This is consistent with a real-world scenario in which unknown files are analyzed based on knowledge learned from past data.

B. Scripts

The Ember dataset developers ran several script programs to analyze the dataset in their previous work [1], and they have publicly distributed these scripts. One script performs a training operation, which transforms input JSON data into feature vectors and then constructs a model to distinguish between the features of malware and those of benign software. The script constructs a model using the LightGBM algorithm [2] and its open-source framework, which is supported by Microsoft and adopted in Windows Defender. Compared with existing algorithms, such as XGBoost [3], LightGBM is faster and consumes less memory, while maintaining almost the same accuracy.

Another script performs a classification operation, which receives unknown program files and computes the malware likelihood scores of the files based on the model created by the first script. The scores range from zero to one, and a file is more likely to be malware if its score is closer to one.

III. Experiments

A. Method

We slightly modified the above-mentioned scripts to extract a subset of features from input data and measure the amount of time elapsed in creating models from training data. We considered features in a feature group as a unit, that is, we selected all the features in a feature group or we selected none. We evaluated 255 combinations of feature groups because 256 combinations could be generated from eight feature groups in total, and one combination was generated by choosing no feature group.

We collected the following information to evaluate each combination:

- Selected feature groups
- Size of input data of the selected feature groups
- Amount of training time to create a model from feature vectors
- Accuracy and false positive rates (FPR) computed by varying the threshold of malware likelihood scores by 0.01. We define accuracy as the ratio of the number of correct answers to the number of all answers, and FPR as the ratio of the number of malware-determination answers to the number of benign samples.

For each combination, we associated a set of feature vectors with a ground-truth label and performed training (model creation) and testing (malware-likelihood computation) operations.

We assigned overall score S_{all} to each combination of feature groups, which was calculated using the following expression:

$$S_{all} = (w_{acc} * S_{acc} + w_{time} * S_{time} + w_{size} * S_{size})/3,$$

where S_{acc}, S_{time}, and S_{size} denote the scores of accuracy, input data size, and training time, respectively. Coefficients w_{acc}, w_{time}, and w_{size} denote the weight of each term, respectively. The scores of accuracy and size are constant in different measurement attempts and on different platforms, whereas the scores of time typically vary. All scores were normalized into the range [0, 1]. The normalized accuracy scores were proportional to their raw values, and those of the data size and training time were inversely proportional to their raw values. The maximum and minimum values of the data size and training time among all value instances were mapped to one and zero, respectively.

TABLE II: Number of Samples in the Dataset

File	Period of collection	Malware	Benign	Unlabeled	Sum
train0	Dec 2006 – Dec 2016	0	50,000	0	50,000
train1	Jan 2017 – Feb 2017	60,000	50,000	60,000	170,000
train2	Mar 2017 – Apr 2017	60,000	50,000	60,000	170,000
train3	May 2017 – Jun 2017	60,000	50,000	60,000	170,000
train4	Jul 2017 – Aug 2017	60,000	50,000	60,000	170,000
train5	Sep 2017 – Oct 2017	60,000	50,000	60,000	170,000
test	Nov 2017 – Dec 2017	100,000	100,000	0	200,000
All		400,000	400,000	300,000	1,100,000

We set the default values of weights w_{acc}, w_{time}, and w_{size} to 2, 1, and 1, respectively. The accuracy scores were weighted twice as much as those of the data size and time because we considered accuracy to be the paramount concern in many scenarios and the scores of size and time tended to be correlated with each other. In this paper, we also show the results obtained when using other combinations of weights: ($w_{acc} = w_{time} = w_{size} = 1$) and ($w_{acc} = 4$, $w_{time} = w_{size} = 1$).

The experiment was conducted in an environment of Ubuntu 18.04.2 running on a computer that had an Intel Xeon E5-2620 processor and 128 GB RAM. The software versions were Python 3.6.7, Ember 0.1.0, LightGBM 2.1.0, scikit-learn 0.19.1, NumPy 1.14.2, and SciPy 1.0.0.

B. Results

1) Overall Score: Table III lists the top 10 combinations in terms of overall score calculated using various combinations of weights. Generally, accuracy values changed according to the adopted malware likelihood threshold. The accuracy values in the table were obtained when the threshold was set to a minimum value that maintained a FPR less than 0.1%. In the following part of this paper, unless stated otherwise, accuracy was calculated using this threshold.

Table IIIa shows the top 10 combinations in terms of the overall scores calculated using the default weights. A combination of general, header, and strings provided the highest overall score, and a combination of general and second provided the second-highest score. The feature groups general, header, section, and strings frequently appeared in the top ranks, and it is natural to conclude that features in these groups were useful in the estimation of malware likelihood. By contrast, the other four groups did not appear and were less likely to be useful, at least under the conditions of this experiment. The best of the four-group combination was the sixth place in the ranking. The accuracy was 87.6% when using four groups, whereas it was 92.7% when using all groups. However, limiting the number of groups to four reduced the data size from 9,327 MB to 2,345 MB (25% of the original), and the training time from 62.78 s to 13.24 s (21.1% of the original). We believe that there certainly exist scenarios in which this level of balance among accuracy, data size, and training time are preferable.

Tables IIIb and IIIc show the top 10 combinations in terms of the overall scores calculated using other weights. When we set all the weights to one (Table IIIb), we obtained a top 10 ranking that was similar to that obtained using the default weights. A remarkable difference was that singletons section and general entered the ranking. These singletons are the feature groups of choice in scenarios in which size and time are important criteria. When the weight of accuracy is increased to four (Table IIIc), the trend of the top 10 ranking slightly varied; the feature groups histogram and byteentropy began to enter the ranking. We consider that these feature groups should be taken into account in scenarios in which the accuracy criterion is by far the most important.

2) Accuracy: We also investigated the top combinations in terms of accuracy alone because we surmised that accuracy is the only criterion in some scenarios. Table IV lists the top 10 combinations of feature groups in terms of accuracy alone. The highest accuracy was obtained when all feature groups were combined, and accuracy values close to the maximum could be obtained with other combinations in the ranking (the difference in accuracy between rank 1 and 10 was as small as 1.0%). The feature groups header, imports, section, and histogram were adopted in all the top 10 combinations, and naturally, they are considered to be particularly useful as far as the accuracy criterion is concerned. Two of the features, header and section, were also found in the ranking in Table III because of their compactness, whereas the remaining two, histogram and imports, were not found in the ranking because of their large sizes. Note that all data sizes in the ranking in Table IV are over 6 GB, whereas those in Table III are under 2.4 GB.

Table V lists the accuracy of "combinations" composed of a single feature group. Those with a high accuracy were imports, section, and histogram. All of them were also found in all the ranks in Table IV. By contrast, when using exports, the accuracy was significantly low. It is even lower than the accuracy expected through random guesses, in which answers are chosen at random in proportion to the ratio of the number of malware and benign samples (i.e., 0.5 in this study). The training times and data sizes shown in Table V tended to be extremely small because of the use of a single feature group.

3) Best Combination under Constraints: We also investigated the best combinations of feature groups when constraints were imposed on the three evaluation criteria. Here we report the result in three sample scenarios in which a certain threshold was set on one of the criteria:

- If accuracy must be over 90%, the number of candidate

TABLE III: Scores of the Top 10 Combinations of Feature Groups and Combination of all Feature Groups

(a) $w_{acc} = 2$, $w_{time} = 1$, $w_{size} = 1$ (default)

Selected feature groups	Number of features	Accuracy (%)	Training time (s)	Data size (MB)	Overall score
general, header, strings	176	83.7	9.12	1,562	0.845
general, section	265	77.0	5.31	1,092	0.840
general, strings	114	75.9	7.54	942	0.830
general, section, strings	369	83.8	12.09	1,725	0.830
header, strings	166	77.9	7.38	1,373	0.829
general, header, section, strings	431	87.6	13.24	2,345	0.827
general, header, section	327	78.7	6.57	1,712	0.826
header, section, strings	421	85.9	12.59	2,155	0.826
section, strings	359	80.4	10.24	1,535	0.825
header, section	317	75.3	5.70	1,523	0.818
All	2,351	92.7	62.78	9,327	0.464

(b) $w_{acc} = 1$, $w_{time} = 1$, $w_{size} = 1$

Selected feature groups	Number of features	Accuracy (%)	Training time (s)	Data size (MB)	Overall score
general, section,	265	76.7	5.31	1,092	0.863
general, strings	114	75.9	7.54	942	0.853
general, header,	72	68.6	3.72	929	0.849
general, header, strings	176	83.7	9.12	1,562	0.848
header, strings	166	77.9	7.38	1,373	0.845
section	255	68.2	4.64	902	0.844
header, section	317	75.3	5.70	1,523	0.840
general, header, section	327	78.7	6.57	1,712	0.839
general	10	56.0	2.67	309	0.836
section, strings	359	80.4	10.24	1,535	0.832
All	2,351	92.7	62.78	9,327	0.309

(c) $w_{acc} = 4$, $w_{time} = 1$, $w_{size} = 1$

Selected feature groups	Number of features	Accuracy (%)	Training time (s)	Data size (MB)	Overall score
general, header, section, strings	431	87.6	13.24	2,345	0.843
general, header, strings	176	83.7	9.12	1,562	0.842
header, section, strings	421	85.9	12.59	2,155	0.837
general, section, strings	369	83.8	12.09	1,725	0.832
general, header, histogram	328	86.5	17.37	2,304	0.826
general, header, byteentropy	328	84.2	14.90	2,119	0.820
general, header, histogram, section,	583	88.7	19.44	3,087	0.820
general, header, histogram, strings	432	87.8	18.78	2,937	0.819
section, strings	359	80.4	10.24	1,535	0.818
general, section	265	77.0	5.31	1,092	0.816
All	2,351	92.7	62.78	9,327	0.618

combinations was limited to 46 and the combination to achieve the shortest training time (and smallest data size) was general, header, exports, section, histogram, and byteentropy. The evaluation values in this case were 31.0 s training time, 6156 MB data size, and 90.5% accuracy.

- If the training time must be under 10 s, the number of candidate combinations was limited to 23 and the combination to achieve the highest accuracy was general, header, and strings. The evaluation values in this case were 9.1 s training time, 1562 MB data size, and 83.7% accuracy.
- If the data size must be under 2 GB, the number of candidate combinations was limited to 20 and the combination to achieve the highest accuracy was general, section, and strings. The evaluation values in this case were 12.1 s training time, 1725 MB data size, and 83.8% accuracy.

4) Discussion: In the following, we discuss each feature based on the experimental data.

- general: This group provided the best of data size scores, and the overall scores when adopting this feature group were excellent. However, it was not useful in terms of accuracy. The accuracy for a single group shown in Table V was not high, and it did not exceed 90% until the threshold of malware likelihood was raised to 0.27 or above. This implies that benign samples were widely

TABLE IV: Top 10 Combinations of Feature Groups in Terms of Accuracy ($w_{acc} = 1$, $w_{time} = 0$, $w_{size} = 0$)

Selected feature groups	Number of features	Accuracy (%)	Training time (s)	Data size (MB)	Overall score
header, imports, section, histogram, general, exports, byteentropy, strings	2,351	92.7	62.78	9,327	0.927
header, imports, section, histogram, general, strings	1,967	92.3	43.63	6,258	0.923
header, imports, section, histogram, general, byteentropy, strings	2,223	92.2	60.62	7,448	0.922
header, imports, section, histogram, byteentropy, strings	2,213	92.2	60.01	7,258	0.922
header, imports, section, histogram, general, exports, byteentropy	2,247	92.0	54.92	8,695	0.920
header, imports, section, histogram, strings	1,957	92.0	46.08	6,069	0.920
header, imports, section, histogram, exports, byteentropy	2,237	91.9	57.14	8,505	0.919
header, imports, section, histogram, general, exports, strings	2,095	91.9	47.27	8,138	0.919
header, imports, section, histogram, byteentropy	2,109	91.8	51.64	6,625	0.918
header, imports, section, histogram, general, exports	1,991	91.7	44.99	7,505	0.917

TABLE V: Ranking of Singleton Feature Groups in Terms of Accuracy

Selected feature group	Accuracy (%)	Training time (s)	Data size (MB)	Overall score
imports	77.8	23.08	2,658	0.778
section	68.2	4.64	902	0.682
histogram	68.1	15.26	1,494	0.681
byteentropy	61.8	14.60	1,309	0.618
strings	61.4	6.85	752	0.614
general	56.0	2.67	309	0.560
header	52.9	3.70	740	0.529
exports	17.2	2.27	1,999	0.172

distributed in the range 0.0–0.3 malware likelihood. The accuracy was highest for a threshold around 0.4, and then accuracy gradually decreased as the threshold increased. This implies that malware samples were widely distributed in the range 0.4–1.0.

- header: This group was useful, particularly when combined with other groups. It was frequently found in the rankings of overall scores and accuracy. However, it did not necessarily provide high accuracy when used alone. In terms of data sizes and training time, it was second only to the general feature group.
- imports: This group appeared in all ranks in the accuracy ranking and provided the best accuracy when used alone. *Therefore, it is the first choice to consider from the accuracy viewpoint.* However, data sizes in this group were large, which prevented it from being high in the overall-score ranking.
- exports: *This group was by far the least useful, and sometimes even harmful.* The accuracy it provided was extremely low and the data sizes were medium. The accuracy and overall scores in some cases were degraded by the adoption of this group. However, it was not completely useless because the scores were sometimes improved by adding it to an adopted combination.
- section: This group was useful and thus frequently appeared in the rankings of overall scores and accuracy. It provided excellent accuracy scores and moderate data size scores.
- histogram: This group was useful because of its high accuracy. However, it was not frequently found in the high ranks of overall scores because of its large data sizes.
- byteentropy: This group was also useful, but not significantly outstanding. It was only moderately useful because of its large data sizes and medium accuracy. It tended toward lower overall scores and did not appear in the overall-score rankings shown in Table IIIa or IIIb.
- strings: This group (and the general group) most frequently appeared in the top 10 of the overall-score ranking. Although it did not provide either the best accuracy or the smallest data size, it provided relatively good values for both of them and balanced the overall scores.

IV. Related Work

The developers of the Ember dataset conducted experiments for distinguishing between malware and benign software based on ML using a combination of all features [1]. The detection accuracy of their method was 92.2% when the FPR was less than 0.1%. Based on their work, we attempted to identify a good subset of features, and found out that 83.7% accuracy could be achieved using only 176 features from three feature groups, whereas they used 2,351 features from eight feature groups.

Pham et al. [4] also used the Ember dataset and measured the detection rate, false alarm rate, and training time using the LightGBM framework. Their results showed that their method could achieve over 97% detection rate, with a 0.1% false alarm rate. They applied feature selection to the input data and decreased the feature dimension by 30%, to reduce the training time but still achieve a better result. However, they did not clarify useful feature sets or present experimental data on trade-offs between feature dimensions and detection rates. There have been several other studies evaluating detection accuracy using the Ember dataset [5], [6], but none have evaluated the usefulness of individual features.

Martín et al. [7] presented a method to detect Android malware programs based on their indirect features and metadata. Examples of information were the permissions, names of the developer and certificate issuer, and number of days on the market. As in our study, they determined the most useful features out of 535 features, and demonstrated that they could identify malware accurately with no more than 15 features. Unlike their study, our study considered Windows programs,

which have a quite different set of features from Android programs.

Saini et al. [8] also presented and evaluated a malware classification method that extracts static features from Windows program files and uses them in ML to distinguish malware programs from other programs. Extracted features include the suspicious section count and function call frequency. They evaluated only a few features and their combinations, whereas, in the present study, we evaluated all combinations of eight groups of more extensive features.

Markel et al. [9] also evaluated an ML-based method for malware classification, which extracted metadata from Windows PE32 files and used them to create a classifier to distinguish between malware programs and benign programs. They extracted 44 features from the headers and two Boolean features from the section entropies, and demonstrated that the header information enabled accurate malware classification. They evaluated the cases of using a single feature alone and identified the most discriminatory features. However, they did not evaluate the case of using a subset of the feature set. Additionally, their target features were mostly those included in the header of program files, and their variety was narrower than that of features available in the Ember dataset.

PE-Miner [10] is a malware detection system that extracts distinguishing features from Windows PE files and performs classification between benign and malware programs based on a data mining algorithm. They evaluated which features' set was the best and demonstrated that the PE-miner approach provided better classification accuracy than an approach using the string feature and an approach by Kolter et al. [11]. Unlike our study, their study did not evaluate the usefulness of other feature combinations or individual features in PE files.

Wang et al., [12] presented a malware detection method using the SVM algorithm for feature selection and malware classification. They demonstrated that their method could achieve high detection accuracy even when they adopted only a small number of PE header features. Unlike our study, their study did not clarify useful features for classification and did not evaluate execution times.

Ye et al. [13] presented a malware detection system that creates classification rules from API call information embedded in Windows PE files, using an object-oriented association mining technique. Unlike our study, they conducted no evaluation of features other than API calls.

V. Conclusion

In this study, we evaluated selected combinations of feature groups of the Ember dataset in terms of the integrated score of classification accuracy, training data size, and elapsed time for creating classification models from training data. We identified useful combinations that achieved a good balance between these criteria, and identified feature groups that achieved high accuracy when used alone. Useful feature groups when used in combination were string information, general file information, header information, and section information. Byte histogram and byte-entropy histogram began to be taken into account when the accuracy standard was significantly important. The singleton feature group with the highest accuracy was imported functions. The feature groups of byte histogram, byte-entropy histogram, and exported functions were less useful, at least from the criteria in this study.

Several directions can be taken for future work. First, it is necessary to evaluate features with a finer granularity. In this study, we evaluated features at the granularity of feature groups, and multiple features of a single group were considered as a single cluster. However, the total number of features was 2,351 and evaluating all combinations of them is impractical, in contrast to the evaluation in this study. It is not trivial to select promising combinations of features, and effective methods for the selection need to be investigated. Next, the usefulness of individual features should be further evaluated on other malware data or other algorithms for detecting malware. For example, similar experiments using non-ML statistical methods, such as chi-square test and Fisher information, should also be conducted. Although this study demonstrated useful features in the context of the Ember dataset and the LightGBM algorithm, it remains unclear whether such features are still useful in other settings. Additionally, we did not exclude a potential bias in the Ember dataset programs. The dataset may possibly contain many variants from some malware families and only a few variants from others. Collecting evaluation results from a wide variety of program data is required to extract findings with generality.

Acknowledgment

We appreciate the feedback provided by Yosuke Chubachi. This work was partly supported by JSPS KAKENHI under Grant Number 17K00179.

References

[1] H. S. Anderson and P. Roth, "EMBER: An open dataset for training static PE malware machine learning models," *ArXiv e-prints*, 2018.

[2] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 3149–3157.

[3] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[4] H.-D. Pham, T. D. Le, and T. N. Vu, "Static PE malware detection using gradient boosting decision trees algorithm," in *Proceedings of the 5th International Conference on Future Data and Security Engineering*, 2018, pp. 228–236.

[5] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, "Robust intelligent malware detection using deep learning," *IEEE Access*, vol. 7, pp. 46 717–46 738, 2019.

[6] C. Wu, J. Shi, Y. Yang, and W. Li, "Enhancing machine learning based malware detection model by reinforcement learning," in *Proceedings of the 8th International Conference on Communication and Network Security*, 2018, pp. 74–78.

[7] I. Martín, J. A. Hernández, A. Muñoz, and A. Guzmán, "Android malware characterization using metadata and machine learning techniques," *Security and Communication Networks*, vol. 2018, no. 5749481, 2018.

[8] A. Saini, E. Gandotra, D. Bansal, and S. Sofat, "Classification of PE files using static analysis," in *Proceedings of the 7th International Conference on Security of Information and Networks*, 2014.

[9] Z. Markel and M. Bilzor, "Building a machine learning classifier for malware detection," in *Proceedings of the 2nd Workshop on Anti-malware Testing Research*, 2014.

[10] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq, "PE-Miner: Mining structural information to detect malicious executables in realtime," in *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, 2009, pp. 121–141.

[11] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004, pp. 470–478.

[12] T.-Y. Wang, C.-H. Wu, and C.-C. Hsieh, "Detecting unknown malicious executables using portable executable headers," in *Proceedings of the Fifth International Joint Conference on INC, IMS and IDC*, 2009, pp. 278–284.

[13] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang, "An intelligent PE-malware detection system based on association mining," *Journal in Computer Virology*, vol. 4, no. 4, pp. 323–334, 2008.